*Softwaretechnik / Software-Engineering*

# *Lecture 4: Software Project Management*

*2018-04-30*

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany



## *Topic Area Project Management: Content*

- VL 2 — **Software Metrics**
  - Properties of Metrics
  - Scales
  - Examples
- VL 3 — **Cost Estimation**
  - "(Software) Economics in a Nutshell"
  - Expert's Estimation
  - Algorithmic Estimation
- VL 4 — **Project Management**
  - Project
  - Process and Process Modelling
- VL 5
  - Procedure Models
  - Process Models
- **Process Metrics**
  - CMMI, Spice

# Content





# Project

# Vocabulary: Project

**project** – A **temporary activity** that is characterized by **having**

- a **start date**,
- specific **objectives and constraints**,
- established **responsibilities**,
- a **budget and schedule**, and
- a **completion date**.

If the objective of the project is to develop a software system, then it is sometimes called a **software development project** or **software engineering project**.

**R. H. Thayer** (1997)

We could refine our earlier definition as follows: a project is **successful** if and only if

- **started** at start date,
- **achieved** objectives,
- **respected** constraints,
- **adheres** to budget and schedule,
- **stops** at completion date.

Whether, e.g., objectives have been achieved can still be **subjective** (→ customer/user happy).



# Vocabulary: Software Project

**(Software) Project** – Characteristics:

- **Duration** is limited.
- Has an **originator** (person or institution which initiated the project).
  - The **project owner** is the originator or its representative.
  - The **project leader** reports to the project owner.
- Has a **purpose**, i.e. pursues a bunch of goals.
  - The most important goal is usually to create or modify software; this software is thus the result of the project, the **product**.
    Other important goals are extension of know-how, preparation of building blocks for later projects, or utilisation of employees.

  The project is called **successful** if the goals are reached to a high degree.
- Has a **recipient** (or will have one).
  - This recipient is the **customer**.
  - Later **users** (conceptionally) belong to the customer.
- **Links people**, **results** (intermediate/final products), and **resources**.

  The **organisation** determines roles of and relations between peoples/results/resources, and the **external interfaces** of the project.

**Ludewig & Lichter** (2013)

# *Project Management*



## *Goals and Activities of Project Management*

- **Main and general goal**:

  Have a **successful** project, i.e. the project **delivers**

  - defined **results**
  - in demanded **quality**
  - within scheduled **time**
  - using the assigned **resources**.

  There may be **secondary goals**, e.g.,

  - build or strengthen good **reputation** on market,
  - acquire **knowledge** which is useful for later projects,
  - develop **re-usable components** (to save resources later),
  - be attractive to **employees**.
  - …

Developer Customer

**software delivery**

- Main **project management activities** (and **responsibilities** of project manager):
  - **Planning**
  - **Assessment and Control**
  - **Recognising and Fighting Difficulties as Early as Possible**
  - **Communication**
  - **Leading and Motivation of Employees**
  - **Creation and Preservation of Beneficial Conditions**

# Common Activities of Project Management

**Without plans, a project cannot be managed.**
Note: mistakes in planning can be hard to resolve.

- **Planning**
- **Assessment and Control**
- **Recognising and Fighting Difficulties as Early as Possible**

Work results and project progress have to be **assessed** and **compared to the plans**; it has to be observed whether participants stick to agreements.

Unforeseen difficulties and problems in projects are not exceptional but usual.

Therefore, project management needs to constantly "screen the horizon for icebergs", and, when spotting one, react timely and effectively.

In other words: **systematic risk management**.

**Distribute information** between project participants (project owner, customer, developers, administration).

- **Communication**
- **Leading and Motivation of Employees**
- **Creation and Preservation of Beneficial Conditions**

**Leading means**: going ahead, showing the way, "pulling" the group.

Most developers want to achieve good results, yet need orientation and feedback (negative and positive).

Provide necessary **infrastructure** and **working conditions** for developers

(against: demanding customers, imprecisely stated goals, organisational restructuring, economy measures, tight office space, other projects, etc.).





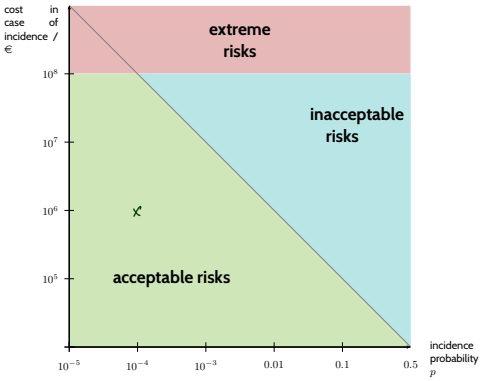
# Quick Excursion: Risk and Riskvalue

**risk** – a problem, which did not occur yet, but on occurrence threatens important project goals or results. Whether it will occur, cannot be surely predicted.

**Ludewig & Lichter (2013)**

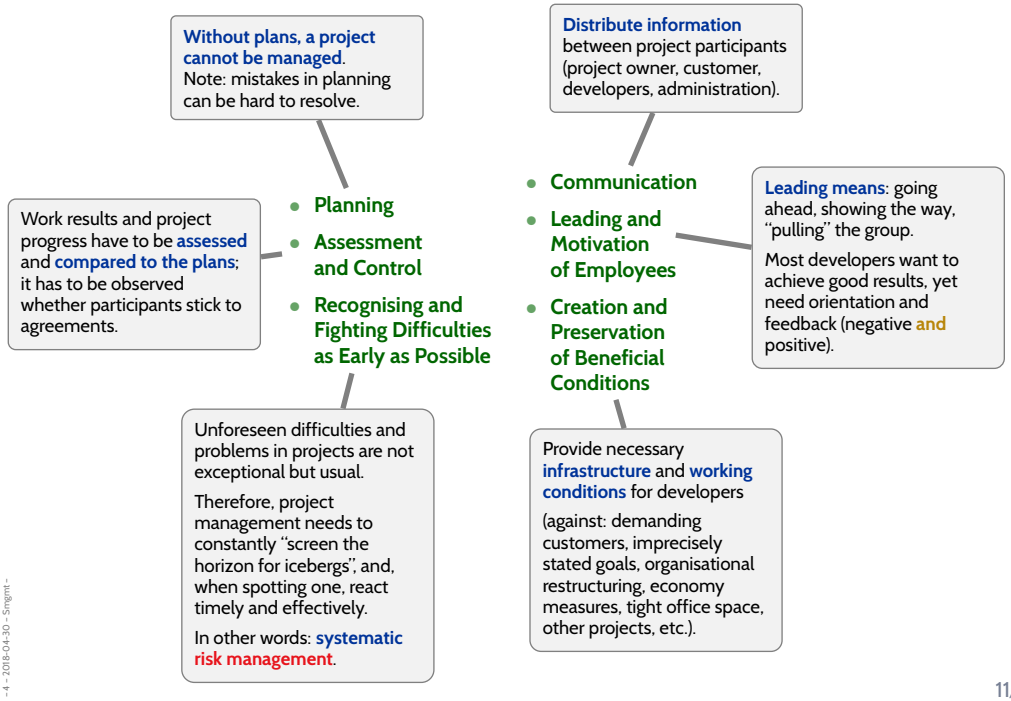
$$\text{riskvalue} = p \cdot K$$

$p$: probability of problem occurrence,
$K$: cost in case of problem occurrence.




- **Avionics** requires: "Average Probability per Flight Hour for Catastrophic Failure Conditions of $10^{-9}$ or 'Extremely Improbable'" (AC 25.1309-1).
- "problems with $p = 0.5$ are not risks, but environment conditions to be dealt with"

# Common Activities of Project Management






# Content

# *Software Development Process*





## *Process*

**Process** –

(1) A sequence of steps performed for a given purpose; for example, the software development process.

(2) See also: task; job.

(3) To perform operations on data.

**IEEE 610.12** (1990)

**Software Development Process** –
The process by which user needs are translated into a software product. The process involves **translating** user needs into **software requirements**, **transforming** the software requirements into **design**, **implementing** the design in **code**, **testing** the code, and sometimes, **installing and checking out** the software for **operational use**.

**IEEE 610.12** (1990)

- The process of a software development project may be
  - implicit,
  - informally agreed on, or
  - explicitly prescribed (by a **procedure** or **process model**).
- **Note**: each software development project **has** a process!

## *Describing Software Development Processes*

Over time, the following **notions** proved useful to describe and model ($\rightarrow$ in a minute) software development processes:

- **role** – has responsibilities and rights, needs skills and capabilities.
  In particular: has responsibility for **artefacts**, participates in **activities**.
- **artefact** – all documents, evaluation protocols, software modules, etc., all products emerging during a development process.
  Is processed by **activities**, may have **state**.
- **activity** – any processing of artefacts, manually or automatic; solves tasks.
  Depends on **artefacts**, creates/modifies **artefacts**.



## *The Concept of Roles*

In a software project, at each point in time,
there is a set $R$ of (active) **roles**, e.g. $R = \{\boxed{\text{mgr}}, \boxed{\text{prg}}, \boxed{\text{tst}}, \boxed{\text{ana}}\}$.

A role has **responsibilities** and **rights**, and necessary skills and capabilities.

**For example**,

- mgr: project manager
  - has the **right** to raise issue reports
  - is **responsible** for closing issue reports
- prg: programmer
  - has the **right** to change the code
  - is **responsible** for reporting unforeseen problems to the project manager
  - is **responsible** for respecting coding conventions
  - is **responsible** for addressing issue reports
- tst: test engineer
  - has the **right** to raise issue reports
  - is **responsible** for quality control

# The Concept of Roles Cont'd

Given a set $R$ of roles, e.g. $R = \{$ mgr, prg, tst, ana $\}$,

and a set $P$ of people, e.g. $P = \{$  $\}$, each with **skills** or **capabilities**.

An aspect of project management is to assign (a set of) people to each role:

$$assign : R \to 2^P$$

powerset of P

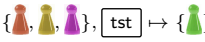
such that each person $p \in assign(r)$ assigned to role $r$
has (at least) the skills and capabilities required by role $r$.

**Note**: $assign$ may change over time, there may be different assignments for different phases.

**Sanity check**: ensure that $assign(r) \neq \emptyset$ for each role $r$.

- **Example**:


one person, one role | multiple persons, one role | one person, multiple roles

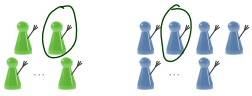
$assign = \{$ mgr $\mapsto \{$  $\}$, prg $\mapsto \{$  $\}$, tst $\mapsto \{$  $\}$, ana $\mapsto \{$  $\}\}$





# Useful and Common Roles

Customer | Developer



Clients | Software people

**Recall**: **roles** "Customer" and "Developer" are assumed by **legal persons**, which often represent many people.

The same legal person may act as "Customer" and "Developer" in the same project.

**Useful and common roles in software projects:**

- **customer**, **user**
- **project manager**
- **(sytems) analyst**
- **software architect**, **designer**
- (**lead**) **developer**
  **programmer**, **tester**, …
- **maintenance engineer**
- **systems administrator**
- **invisible clients**: legislator, norm/standard supervisory committee

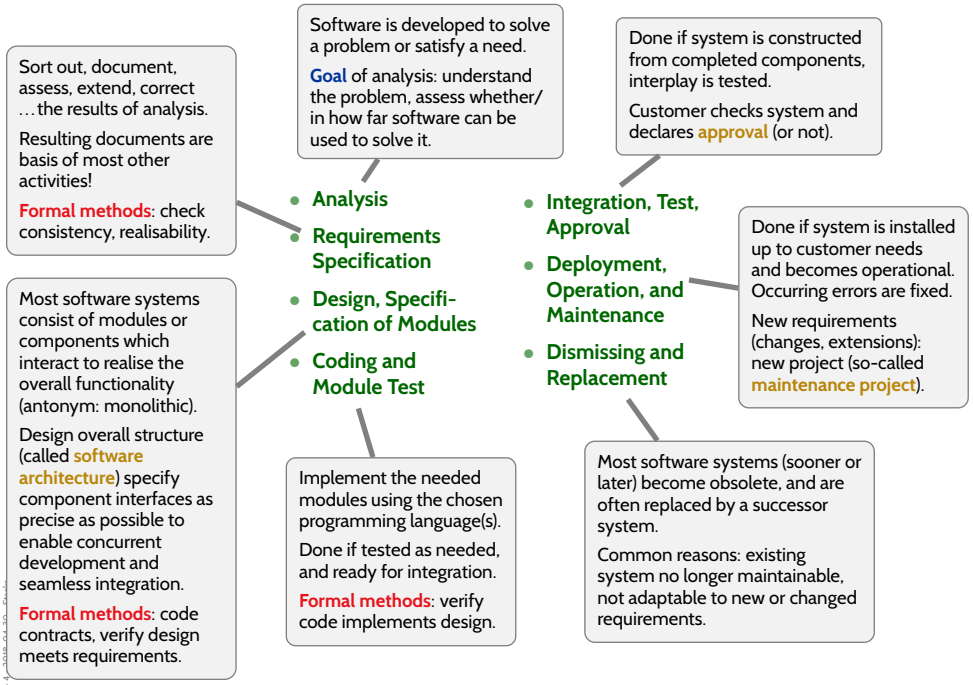
# Describing Software Development Processes

Over time, the following **notions** proved useful to describe and model (→ in a minute) software development processes:

- **role** – has resposibilities and rights, needs skills and capabilities.
  In particular: has responsibility for **artefacts**, participates in **activities**.
- **artefact** – all documents, evaluation protocols, software modules, etc., all products emerging during a development process.
  Is processed by **activities**, may have **state**.
- **activity** – any processing of artefacts, manually or automatic; solves tasks.
  Depends on **artefacts**, creates/modifies **artefacts**.



# Common Activities in Order to Develop or Adapt Software



# Describing Software Development Processes

Over time, the following **notions** proved useful to describe and model (→ in a minute) software development processes:

- **role** – has resposibilities and rights, needs skills and capabilities.
  In particular: has responsibility for **artefacts**, participates in **activities**.
- **artefact** – all documents, evaluation protocols, software modules, etc., all products emerging during a development process.
  Is processed by **activities**, may have **state**.
- **activity** – any processing of artefacts, manually or automatic; solves tasks.
  Depends on **artefacts**, creates/modifies **artefacts**.



# Phases, Milestones

> A **phase** is a continuous, i.e. not interrupted range of time in which certain works are carried out and completed. At the end of each phase, there is a **milestone**.
>
> A phase is **successfully completed** if the criteria defined by the milestone are satisfied.
>
> **Ludewig & Lichter** (2013)

- Phases (in this sense) **do not overlap!**
  Yet there may be different "threads of development" running in parallel, structured by different milestones.
- Splitting a project into phases **makes controlling easier**;
  milestones may involve the customer (accept intermediate results) and trigger payments.
- The **granularity** of the phase structuring is critical:
  - very short phases may not be tolerated by a customer,
  - very long phases may mask significant delays longer than necessary.

  **If necessary**:
  define **internal** (customer not involved) and **external** (customer involved) milestones.

## *Milestones, Deadlines*

> A **phase** is a continuous, i.e. not interrupted range of time in which certain works are carried out and completed. At the end of each phase, there is a **milestone**.
>
> A phase is **successfully completed** if the criteria defined by the milestone are satisfied.
>
> **Ludewig & Lichter** (2013)

- Whether a milestone is **reached** (or successfully completed) must be **assessable** by
  - clear,
  - objective, and
  - unambiguous

  criteria.
- The **definition of a milestone** often comprises:
  - a definition of the **results** which need to be achieved,
  - the required **quality** properties of these results,
  - the desired **time** for reaching the milestone (the **deadline**), and
  - the instance (person or committee) which **decide**s whether the milestone is reached.
- Milestones can be part of the **development contract**;
  not reaching a defined milestone as planned can lead to **legal claims**.



## *Cycle and Life Cycle*

> **cycle** – (1) A period of time during which a set of events is completed. [...]
>
> **IEEE 610.12** (1990)

> **software development cycle** – The period of time that begins with the **decision to develop** a software product and ends when the software is **delivered**. [...]
>
> **IEEE 610.12** (1990)

> **software life cycle** – The period of time that begins when a software product is **conceived** and ends when the software is no longer **available for use**. [...]
>
> **IEEE 610.12** (1990)

> **system life cycle** – The period of time that begins when a system is **conceived** and ends when it is **no longer available for use**.
>
> **IEEE 610.12** (1990)

## *Software Life and Development Cycle*

**software development cycle** – The period of time that begins with the **decision to develop** a software product and ends when the software is **delivered**.

This cycle typically includes

- a **requirements phase**,
- a **design phase**,
- an **implementation phase**,
- a **test phase**, and
- sometimes an **installation and checkout phase**.

**Notes**:

(1) the phases listed above may overlap or be performed iteratively, depending upon the software development approach used.

(2) This term is sometimes used to mean a longer period of time, either the period that ends when the software is no longer being enhanced by the developer, or the entire software life cycle.

**IEEE 610.12** (1990)

**software life cycle** – The period of time that begins when a software product is **conceived** and ends when the software is no longer **available for use**.

The software life cycle typically includes

- a **concept phase**,
- a **requirements phase**,
- a **design phase**,
- an **implementation phase**,
- a **test phase**,
- an **installation and checkout phase**,
- on **operation and maintenance phase**, and,
- sometimes, a **retirement phase**.

**Note**: These phases may overlap or be performed iteratively. **IEEE 610.12** (1990)



## *Describing Software Development Processes*

Over time, the following **notions** proved useful to describe and model (→ in a minute) software development processes:

- **role** – has resposibilities and rights, needs skills and capabilities.
  In particular: has responsibility for **artefacts**, participates in **activities**.
- **artefact** – all documents, evaluation protocols, software modules, etc., all products emerging during a development process.
  Is processed by **activities**, may have **state**.
- **activity** – any processing of artefacts, manually or automatic; solves tasks.
  Depends on **artefacts**, creates/modifies **artefacts**.

# Content



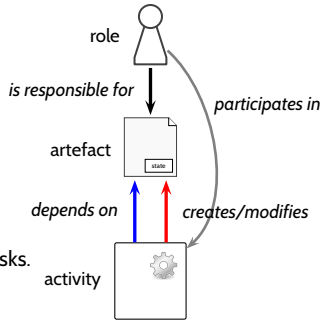


# Software Project Planning

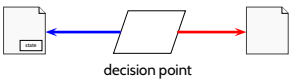
# Describing Software Development Processes

Over time, the following **notions** proved useful to describe and model ($\rightarrow$ in a minute) software development processes:

- **role** – has resposibilities and rights, needs skills and capabilities.
  In particular: has responsibility for **artefacts**, participates in **activities**.
- **artefact** – all documents, evaluation protocols, software modules, etc., all products emerging during a development process.
  Is processed by **activities**, may have **state**.
- **activity** – any processing of artefacts, manually or automatic; solves tasks.
  Depends on **artefacts**, creates/modifies **artefacts**.

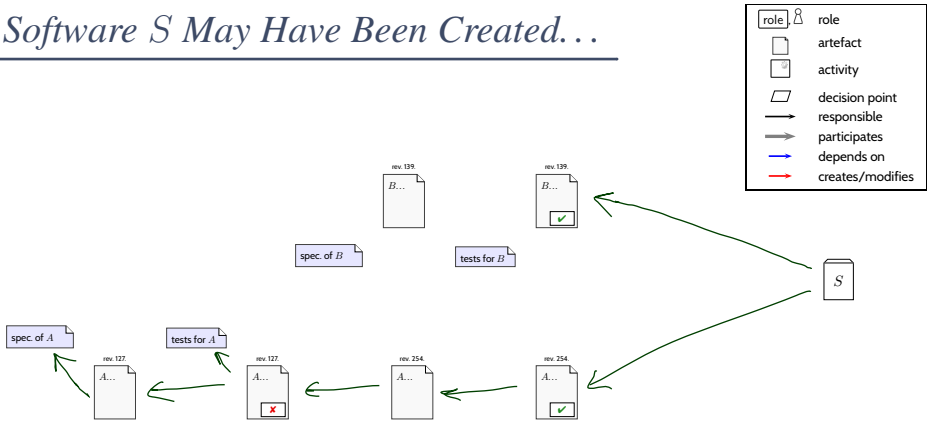



- **decision point** – special case of activity: a decision is made based on **artefacts** (in a certain state), creates a **decision artefacts**.
  Delimits phases, may **correspond to milestone**.

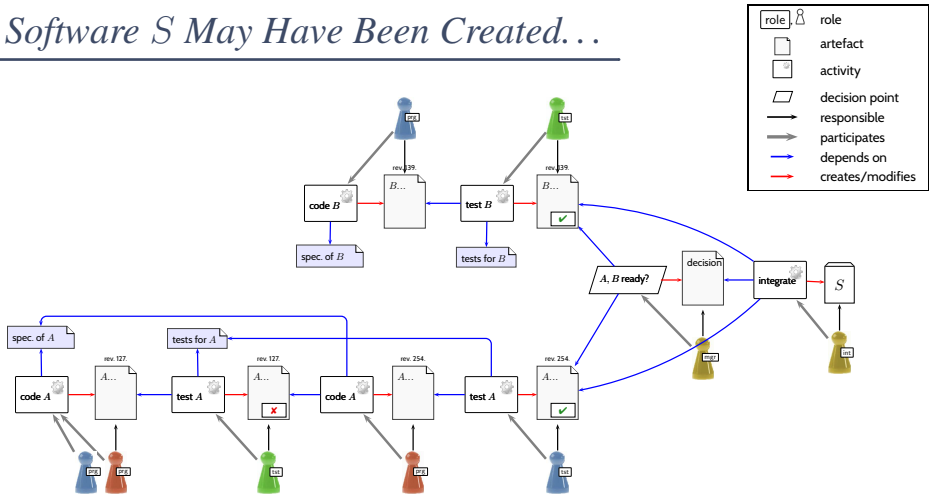





# How Software $S$ May Have Been Created...



# *How Software $S$ May Have Been Created...*

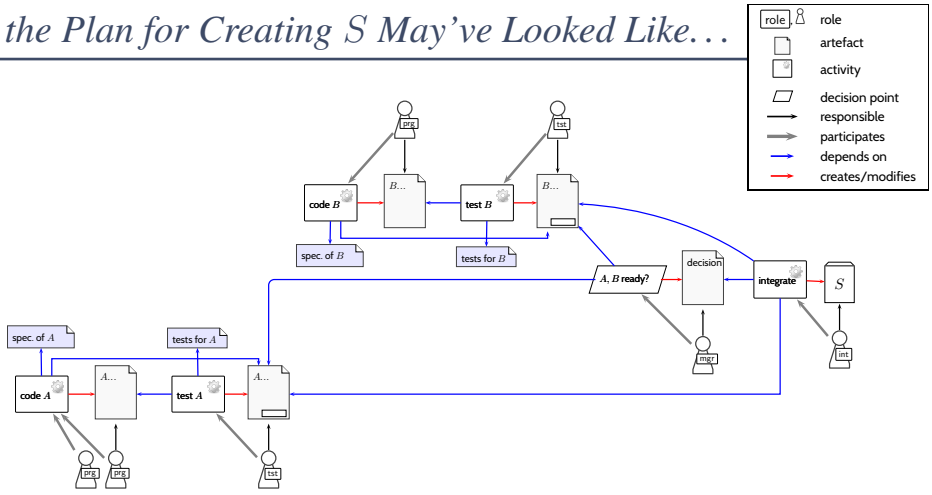



- $S$ consists of modules $A$ and $B$.
- Assume: specifications and test cases for $A$ and $B$ were available.
- Person coded $B$ (according to spec.), then person tested $B$ (with test cases), no errors found.
- Person coded $A$, with the help of person . Then person tested $A$, some errors found.
- Person fixed $A$, person tested again, no errors found.
- $A$ and $B$ ready caused a positive decision, then person integrated $A$ and $B$ and obtained $S$.



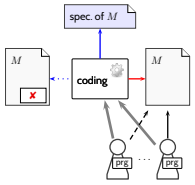


# *How the Plan for Creating $S$ May've Looked Like...*

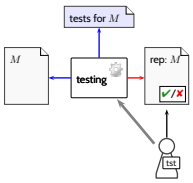



- $S$ consists of modules $A$ and $B$; specifications and test cases for $A$ and $B$ are available.
- Some prg codes $B$ (according to spec.), then some tst tests $B$ (with test cases), and creates test report.
- Some prg codes $A$, with the help of some prg. Then some tst tests $A$, and creates test report.
- If errors in $A$ found, some single prg fixes $A$, some tst tests again, and creates test report.
- If $A$ and $B$ ready causes a positive decision, then some int integrates $A$ and $B$ and obtains $S$.

# How the Plan for Creating $S$ May Have Been Created...



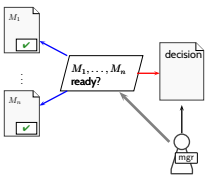

- A **software module** $M$ has a responsible prg, any number of prg may **help** with work on $M$.
- A **software module** $M$ is created/modified by activity **coding**.
- Activity **coding** depends on a **specification** of $M$, and may consider a **positive test report** for $M$.
- The responsible prg (and the helper prg's) participate in activity **coding**.
- Activity **coding** is done, if $M$ exists and there is a negative **test report** for $M$ (all tests passed).



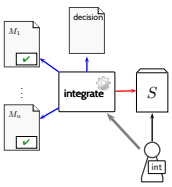

- A **test report** for a module $M$ has a responsible tst.
- A **test report** is created/modified by activity **testing**.
- Activity **testing** depends on **software module** $M$ and **tests** (in state "finished") for $M$.
- The responsible tst participates in activity **testing**.
- Activity **testing** is done, if $M$ exists and there is a negative **test report** for $M$ (all tests passed).



# How the Plan for Creating $S$ May Have Been Created...

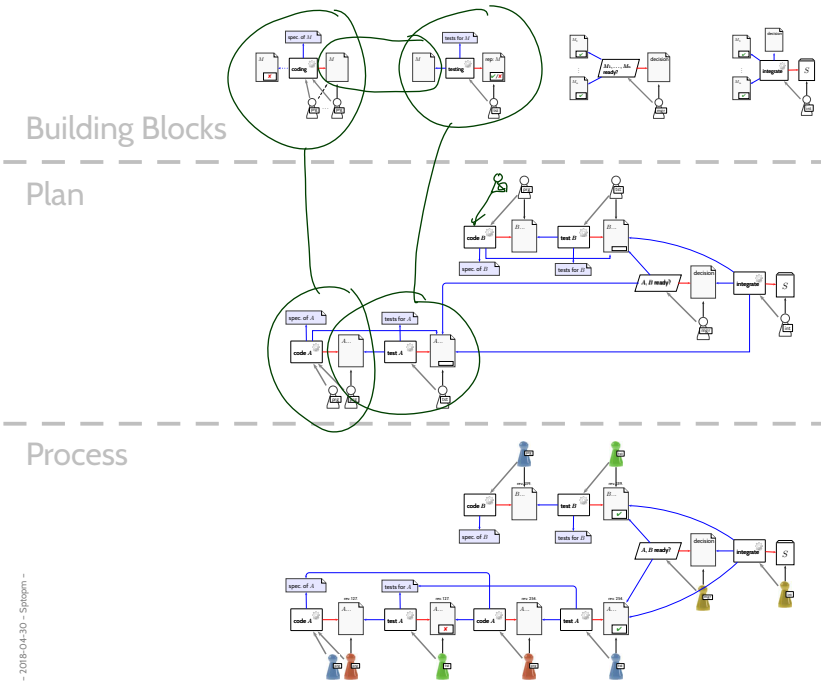



- A **ready decision** for a modules $M_1, \ldots, M_n$ has a responsible mgr.
- A **ready decision** is created/modified by decision point **ready?**.
- Decision point **ready?** depends on negative **test reports** for $M_1, \ldots, M_n$.
- The responsible mgr participates in decision point **ready?**.
- Decision point **ready?** is done, if a positive decision exists.




- A **software** $S$ has a responsible int.
  is created by integrating modules $M_1, \ldots, M_n$
- A **software** is created/modified by activity **integration**.
- Activity **integration** depends on **software modules** $M_1, \ldots, M_n$ in state "finished".
- The responsible int participates in activity **integrate**.
- Activity **integration** is done, if $S$ exists.

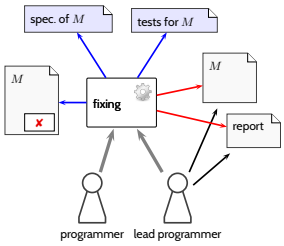
# *From Building Blocks to Process (And Back)*






# *Building Blocks Can Be Arbitrarily Complicated*

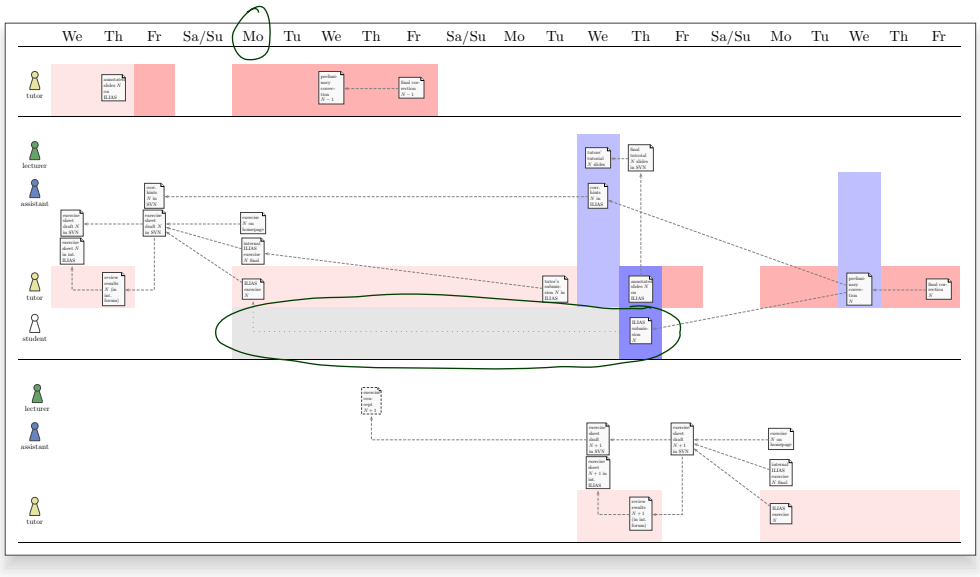
- **Example**: Distinguish **coding** and **fixing** software.




- If there is a negative test result for $M$,
- a leadprogrammer is responsible for fixing $M$,
- the programmer who was responsible for the initial version assists;
- fixing depends on the **test cases**, in addition to the **specifiation** of $M$,
- a **report** is created (analysis of the error, documentation of the fix).

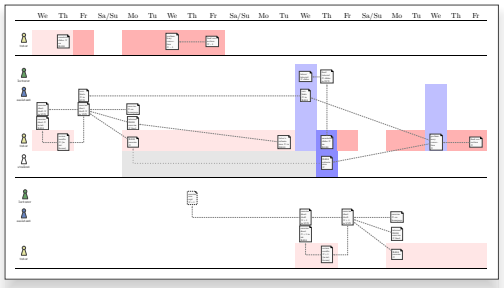
- By using such **building blocks**, the project manager
  - can **prescribe** particular procedures,
  - analyse, which **roles** need to be filled in a project,
  - avoid to "forget" things.

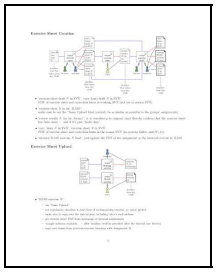
# *By the Way: Process Model of Tutorials*

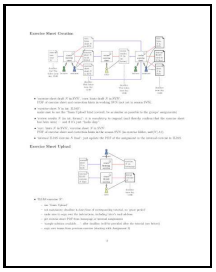


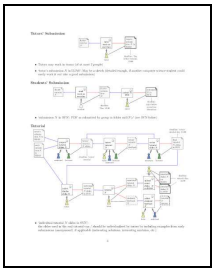


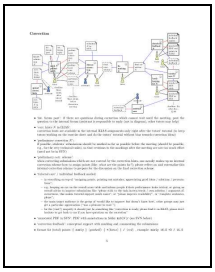

# *By the Way: Process Model of Tutorials*











– 4 – 2018-04-30 – Stuproc –
36/49

# Content





# Process vs. Procedure Models

## *Process Description and Reference Model*

**process description** – documented expression of a set of activities performed to achieve a given purpose.

NOTE: A process description provides
**an operational definition of the major components of a process**.

The description specifies, in a **complete, precise, and verifiable** manner,
the requirements, design, behavior, or other characteristics of a process.

It also may include **procedures**
for **determining** whether these provisions have been satisfied.

Process descriptions can be found at the **activity, project, or organizational level**.

**IEEE 24765** (2010)

**process reference model** – a model comprising definitions of processes in a life cycle described in terms of process purpose and outcomes, together with an architecture describing the relationships between the processes. **IEEE 24765** (2010)



## *Process vs. Procedure Model*

(Ludewig and Lichter, 2013) propose to distinguish: **process model** and **procedure model**.

- A **Process model** ('Prozessmodell') comprises
  - (i) **Procedure model** ('Vorgehensmodell')
    e.g., "waterfall model" (70s/80s).
  - (ii) **Organisational structure** – comprising requirements on
    - project management and responsibilities,
    - quality assurance,
    - documentation, document structure,
    - revision control.

  e.g., V-Modell, RUP, XP (90s/00s).

- In the literature, **process model** and **procedure model** are often used as synonyms; there is not universally agreed distinction.

# Anticipated Benefits of Process Models

- **"economy of thought"**
  – don't re-invent principles.
- **quantification, reproducibility**
  – one can **assess the quality** of **how** products are created (→ CMMI).
  Identify weaknesses, learn from (bad) experience, improve the process.
- **fewer errors**
  – e.g., testing a module cannot be forgotten because the "ready" decision point depends on module with "test passed" flagged.
- **clear responsibilities**
  – fewer "I thought **you**'d fix the module!"

- **Process-model-ing** is easily **overdone** – the best process model is **worthless** if your software people don't "live" it.
- Before introducing a process model
  - understand what you have, understand what you need.
  - process-model as much as needed, not more (→ tailoring).
  - assess whether the new/changed process model makes matters better or worse (→ metrics)
- **Note**: customer may require a certain process model.





# Content

- **(Software) Project**
- **Project Management**
  - Goals
  - Common Activities
  - Excursion: Risk
- **Software Development Processes**
  - Roles, Artefacts, Activities
  - Costs and Deadlines
    - phase, milestone, deadline
    - cycle, life cycle, software life cycle
- **Development Process Modelling**
  - process vs. process model
- **Procedure and Process Models**
  - "Code and Fix"
  - The (infamous) Waterfall Model

## *Tell Them What You've Told Them...*

- **Project**; has (among others)
  - **project owner**, **project leader**
  - **goals** (Excursion: **Risk**)
  - **process** – each project has one
- **processes** can be **modelled**
  - **descriptive** ("we did it like that"), or
  - **prescriptive** ("please to it like that")
- A **process model** relates
  - **roles**, **artifacts**, **activities**, **decision points**
  - relations: **responsibility**, **dependency**, **creation/modification**.
- A process model can allow us to (→ exercises)
  - devise a **schedule** (**who does what when**)
  - estimate and control **phases** and **deadlines**.
- Distinguish **procedure model** and **process model**.
- Example: The **Waterfall** procedure model.



## *References*

# *References*


IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990.

ISO/IEC/IEEE (2010). *Systems and software engineering – Vocabulary*. 24765:2010(E).

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Rosove, P. E. (1967). *Developing Computer-based Information Systems*. John Wiley and Sons.

Thayer, R. H. (1997). *Tutorial – Software Engineering Project Management*. IEEE Society Press, revised edition.